A Discussion of Optimization Strategies and Performance for Unstructured Computations in Parallel HPC Platforms

DOD HPC UGC 2001

June 21, 2001

D. Shires, R. Mohan[*], and A. Mark

Computational and Information Sciences Directorate
High Performance Computing Division

[*]*University of Minnesota*

Outline

- Introduction and Motivation
- Computational Problem
- Parallel Solution Approaches
  - MPI
    - Optimizations
  - HPF
    - Optimizations
- Performance
  - HPF vs. MPI
  - MPI Cross-Platform
- Concluding Remarks

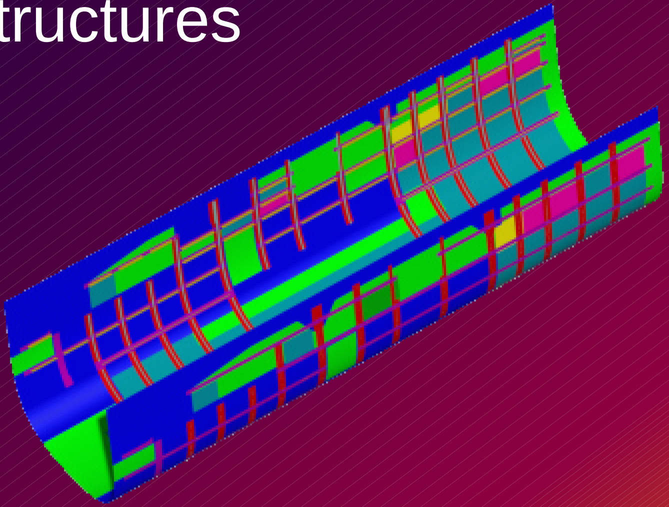
Motivation

- Investigate and explore various approaches to parallelism on DOD HPC systems
  - Portable languages and libraries
  - Data parallelism, message passing
- Perform studies in the context of a real Army (and DOD, and industry, etc.) problem
  - Promote composite material insertion through risk reduction
  - Reduce risk by way of process simulations
  - Use novel solution approaches and parallel formulations for large-scale simulations
  - Assist in fielding Future Combat Systems (FCS)

Applications

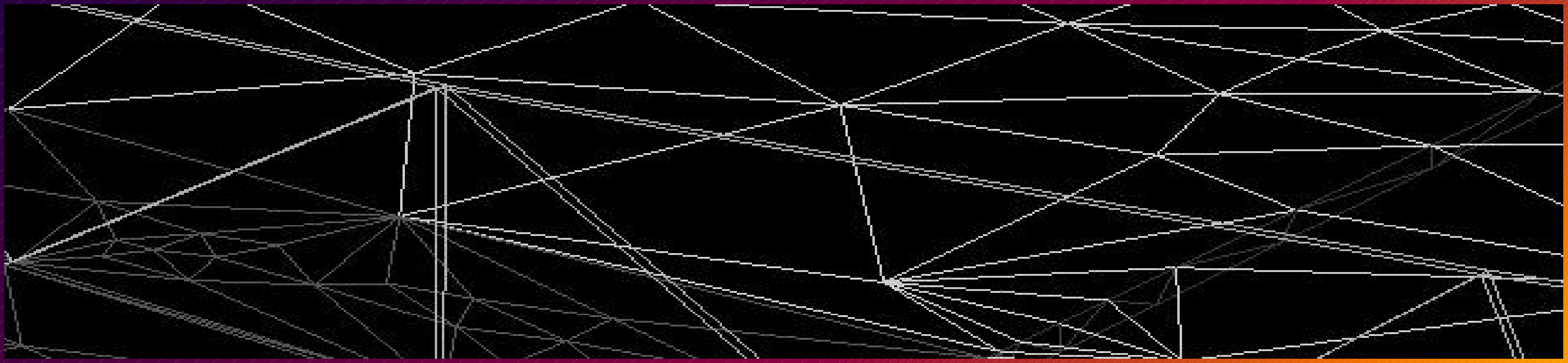
- Ground vehicles

- Rotary wing structures



Computational Problem

- Composite manufacturing prediction of
  - Resin impregnation behavior
  - Track pressure field and flow fronts
- Addresses various liquid composite molding processes
  - RTM, VARTM, Low pressure RTM
- Eulerian fixed-meshes used
- Unstructured meshes represent geometrically complex models
  - Represent difficult challenges to getting good parallel performance



Parallel Software Developments

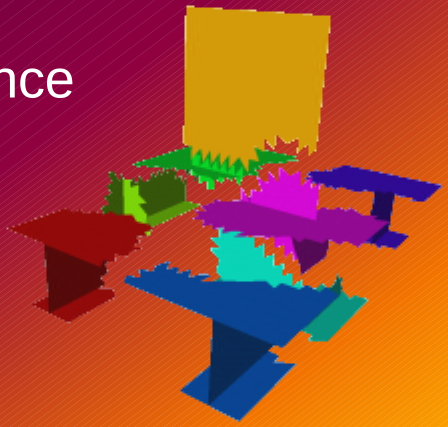
- Software developments performed under CHSSI IMT-4
- Primary deliverable is the MPI-based Composite Manufacturing Process Simulation Environment (COMPOSE)
- Consists of a suite of parallel/serial pre- and post-processing tools
- New modules are continually under development to fill RDEC and industry requirements

HPF and MPI
Pros and Cons

- HPF
  - Higher conceptual level
  - Easy to use directives and data distribution
  - A language; requires robust compilers
  - Compiler controlled communication difficult to optimize
  - Some details are proprietary-obscured
- MPI
  - Low level, “assembly language” parallelism
  - Requires data decomposition
  - Use well-tested native compilers; built on a library
    - Good optimization and performance analysis
  - Tedious attention to detail can give good results

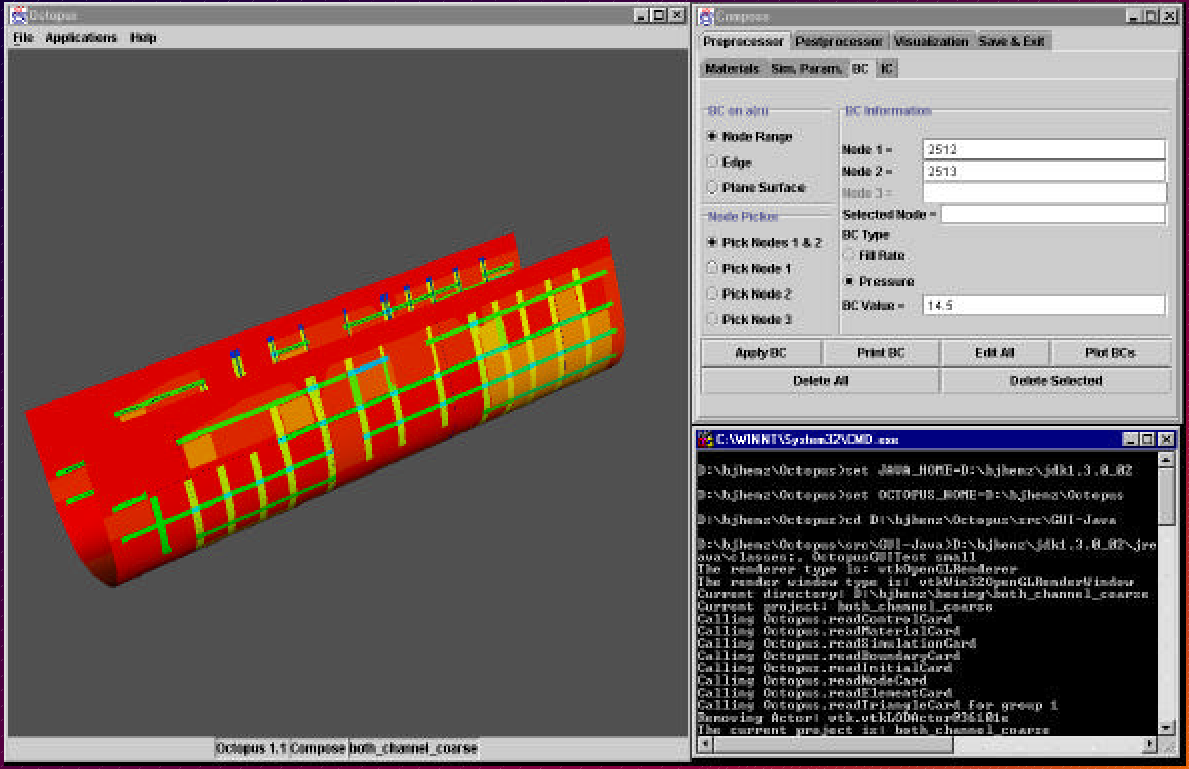
MPI Parallel Software Developments

- Provides 2-D and 3-D solutions
  - Triangular, quadrilateral, and tetrahedral elements
- Preprocessors written in C++
  - `compose_convert`
    - Converts from other formats (NASTRAN) to COMPOSE
  - `compose_check`
    - Checks mesh to ensure mesh connectivity
  - `compose_optimize`
    - Optimizes mesh structure for cache performance
  - `compose_partition`
    - Partitions mesh for multiprocessor execution



MPI Parallel Software Developments

- Pre- and post-processing developments are ongoing in Java
  - Provides cross-platform support
  - Provides better tech transfer to customer base


Octopus
File Applications Help
Octopus 1.1 Compose both_channel_coarse
Compose
Preprocessor Postprocessor Visualization Save & Exit
Materials Sim. Param. BC IC
BC on a/an
Node Picker
BC Information
Apply BC
Print BC
Edit All
Plot BCs
Delete All
Delete Selected
C:\WINNT\System32\CMD.exe

BC on a/an: [✓ Node Range] Edge Plane Surface

Node Picker: [✓ Pick Nodes 1 & 2] Pick Node 1 Pick Node 2 Pick Node 3

Node 1 = 2512

Node 2 = 2513

Node 3 =

Selected Node =

BC Type: Fill Rate [✓ Pressure]

BC Value = 14.5

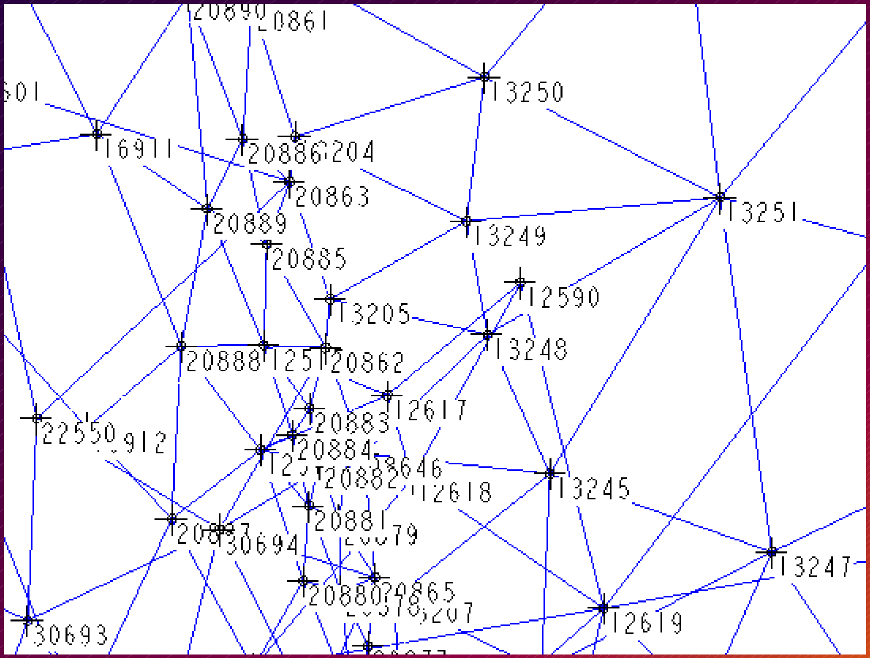
```
D:\bjhenz\Octopus>set JAVA_HOME=D:\bjhenz\jdk1.3.0_02

D:\bjhenz\Octopus>set OCTOPUS_HOME=D:\bjhenz\Octopus

D:\bjhenz\Octopus>cd D:\bjhenz\Octopus\src\GUI-Java

D:\bjhenz\Octopus\src\GUI-Java>D:\bjhenz\jdk1.3.0_02\jre
ava\classes;. OctopusGUITest small
The renderer type is: vtkOpenGLRenderer
The render window type is: vtkWin32OpenGLRenderWindow
Current directory: D:\bjhenz\boeing\both_channel_coarse
Current project: both_channel_coarse
Calling Octopus.readControlCard
Calling Octopus.readMaterialCard
Calling Octopus.readSimulationCard
Calling Octopus.readBoundaryCard
Calling Octopus.readInitialCard
Calling Octopus.readNodeCard
Calling Octopus.readElementCard
Calling Octopus.readTriangleCard for group 1
Removing Actor: vtk.vtkLODActor@361@1e
The current project is: both_channel_coarse
```

MPI Parallel Software Developments

- Core solver written in Fortran 90
  - Slightly more friendly development environment over FORTRAN 77
    - Heap memory
    - Abstract data types
    - Assumed shape arrays with intrinsics
- Message passing characteristics
  - Attempted to remove all explicit barrier calls through control flow analysis
  - Attempted to hide communication behind computation
    - Tight loop structures of solver and update sections limited this somewhat
  - Messages consist of non-blocking sends, blocking receives, and global reduction operations

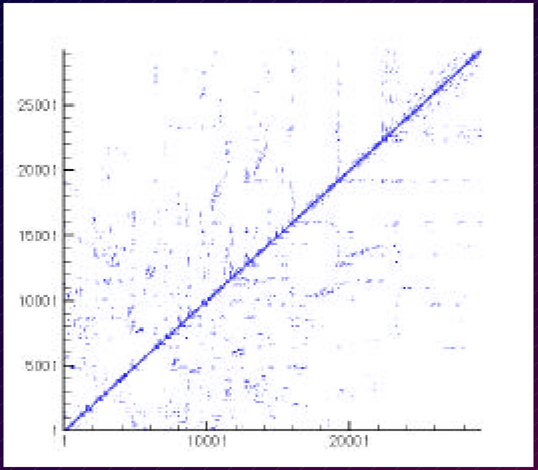
MPI Optimizations

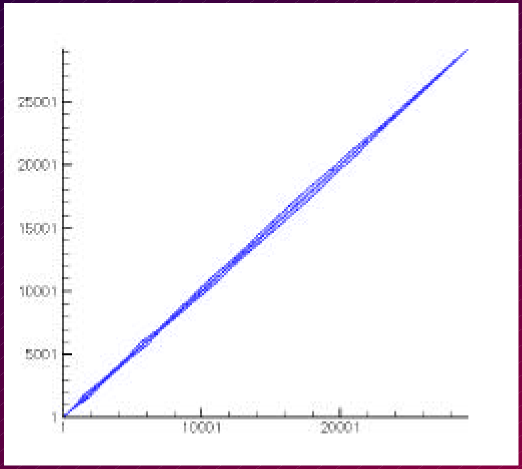
- MPI is SPMD parallelism
  - Sequential optimization provides parallel optimization
- Data optimizations for cache
  - Poorly numbered meshes from CAD packages do little for cache affinity



MPI Optimizations

- Implemented a technique based on Reverse Cuthill-McKee





Distribution of non-zero entries in a finite element sparse matrix

- Effects
  - 10% reduction of wall clock time on T3E-1200
  - Before and after not tested on IBM Nighthawk Power3

MPI Optimizations

- Large cache (8 MB) SGI Origin 3800 showed no wall clock time change
- Hardware counters did show some improvements

Statistic	Original	Renumbered
L1 Cache Line Reuse	5.42	5.67
L2 Cache Line Reuse	4548.95	9530.24
Memory Bandwidth Used (MB/s)	1.21	0.43

MPI Optimizations

- Compiler and source code optimizations
  - Careful profiling can reveal problems
  - One update routine contained a division operation inside a loop

```
#<swps>    2 flops          (  7% of peak)
                        (madds count as 2)
#<swps>    0 madds          (  0% of peak)
#<swps>    3 mem refs       ( 21% of peak)
#<swps>    3 integer ops    ( 10% of peak)
#<swps>    8 instructions ( 14% of peak)
```

  - Used multiply by reciprocal, loop invariant code motion to help pipeliner and instruction scheduler

```
#<swps>    8 flops          ( 30% of peak)
                        (madds count as 2)
#<swps>    4 madds          ( 30% of peak)
#<swps>   13 mem refs       (100% of peak)
#<swps>    6 integer ops    ( 23% of peak)
#<swps>   23 instructions ( 44% of peak)
```

MPI Optimizations

- Final optimization dealt with instruction scheduling
  - Reverse procedure integration
    - Operates as an inverse to inlining
    - Contiguous code segments internal to a subroutine are moved to a separate subroutine
    - Performance analysis and investigation of assembly code revealed complicated instruction scheduling in a critical region
    - Matrix-vector multiply inside a larger loop
    - Compiler generated prefetching instructions for outer loop of mat-vec multiply
    - This technique allows some control (along with compiler options) for phase ordering type problems
      - Prefetching, out of order execution, software pipelining
    - Results:
      - 15% reduction in time for SGI O3K
      - No change on T3E-1200
      - Not tested on IBM Nighthawk Power 3

HPF Parallel Software Developments

- Data parallel model
- New, more robust Portland Group 3.2 compiler came on-line
- Allows for asymmetric block distribution of data
  - Critical for good performance on unstructured grids
  - Overcomes version 3.0 restrictions requiring conformable arrays
  - We encountered problems with interactions between intermediate code and native compilers
  - Not enough time to get everything working
- Past optimizations covered in previous paper/presentation
- Currently testing new compiler and software on SGI Origin 3800 system

Performance

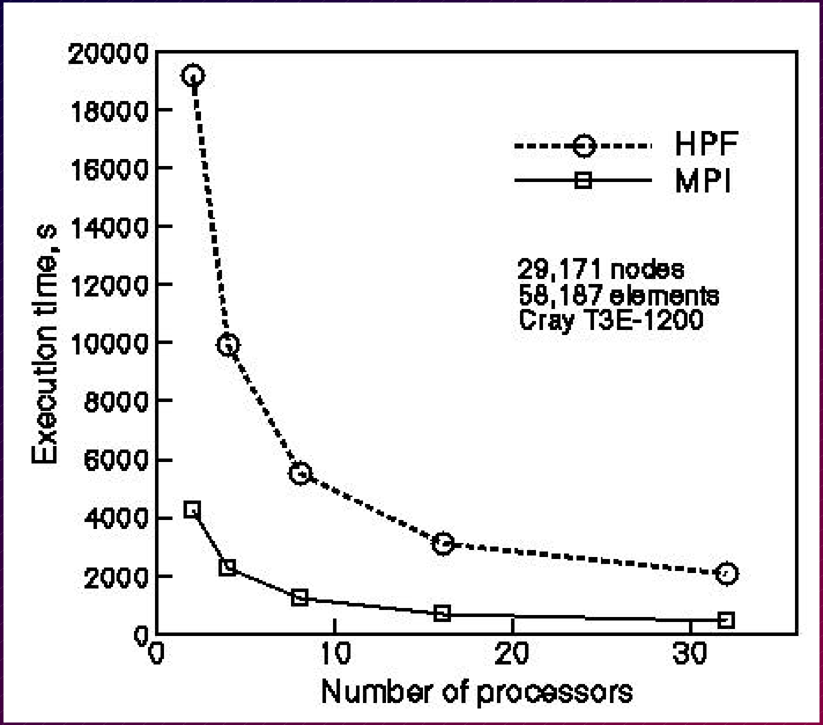
- HPC platforms used include
  - SGI Origin 3800
    - Scalable currently limited to 128 processor system
  - IBM Nighthawk 2 SMP with Power3 nodes
    - Scalability tests up to 512 processors
  - Cray T3E-1200
    - Scalability tests up to 1024 processors

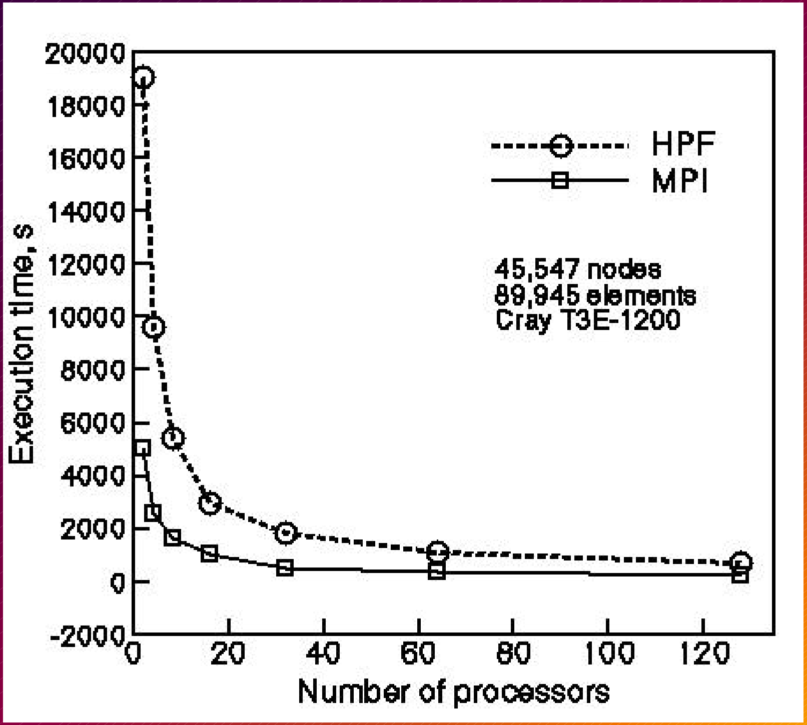
Performance of HPF versus MPI

- Currently limited to Cray T3E system
  - Timing and compiler constraints
  - Past optimization of PGHPF compilers for that system
- Comparable compiler flags
  - PGHPF `–fast` option, invokes native compiler with `–Ounroll –Opipeline2 –Oscalar3` flags
  - MPI Fortran 90 compiled with `–O3,pipeline3` flags
  - Times are exclusive of I/O

Performance of HPF versus MPI

- MPI outperformed HPF in every trial
  - Factors ranged from 2.7 to 4.5 times faster
- HPF meshes were not renumbered using RCM
  - May reduce time by up to 10% as seen with MPI runs

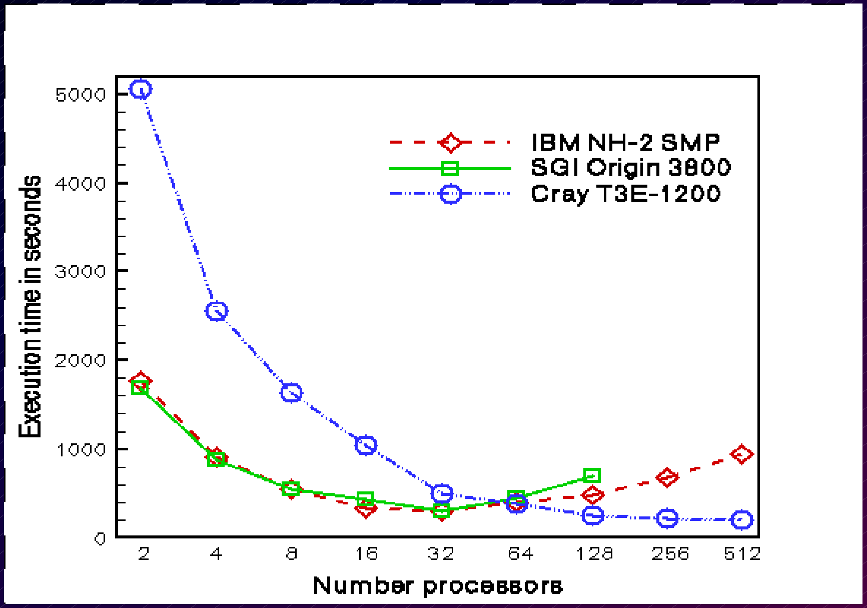


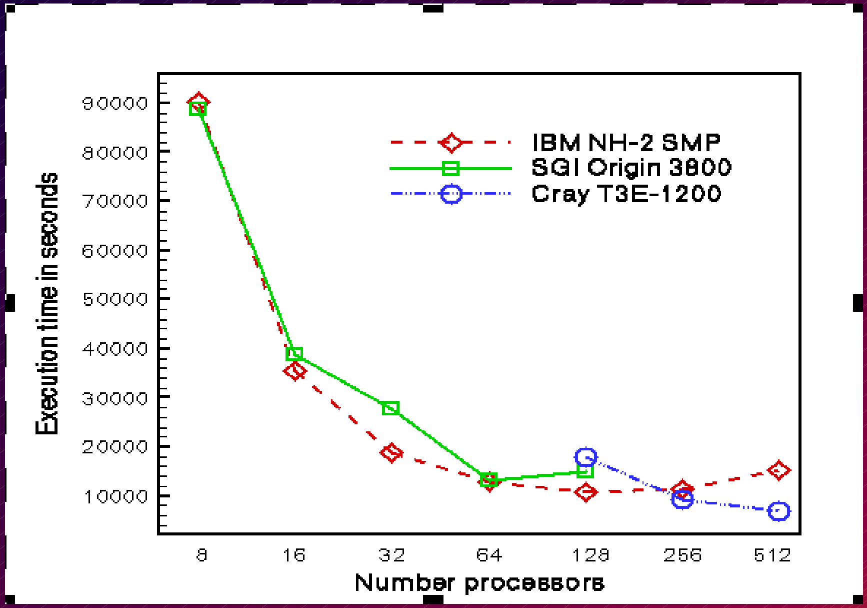


MPI Cross-Platform Performance

- Jobs submitted in standard production queues
  - IBM system in pioneer mode
    - At or near full capacity
- 5 large-scale meshes tested, 2 presented
  - Mesh 1 - 45,547 nodes, 89,945 elements
  - Mesh 2 – 405,327 nodes, 809,505 elements
- Overall timings also dependent on processing conditions, which were different

MPI Cross-Platform Performance





- Performance starts to degrade as code becomes communication bound
- Good performance up to 64 and 128 processors
  - 1 processor 2 week run can complete in about 3.5 hours using 64 PE
- T3E requires roughly double the CPUs to get comparable performance

Concluding Remarks

- Parallel computing and CHSSI support have enabled solutions to large-scale manufacturing problems
  – Applications to ground vehicles, air structures, etc.
- HPF and MPI provide valid solution approaches
  – MPI is more efficient and portable
  – HPF continues to mature, but not fast enough
- Careful performance analysis and profiling can reveal many optimization opportunities
  – Execution time on the SGI 3800 reduced by about 35%